06 | MapReduce（一）：源起Unix的设计思想

2021-10-01 徐文浩

《大数据经典论文解读》 课程介绍 >

讲述：徐文浩
时长 20:02 大小 18.36M

你好，我是徐文浩。

在解读完 GFS 的论文之后，相信你现在对“分布式系统”已经有了初步的了解。本质上，GFS 是对上千台服务器、上万块硬盘的硬件做了一个封装，让 GFS 的使用者可以把 GFS 当成一块硬盘来使用。

通过 GFS 客户端，无论你是要读还是写海量的数据，你都不需要去操心这些数据最终要存储到哪一台服务器或者哪一块硬盘上。你也不需要担心哪一台服务器的网线可能松了，哪一块硬盘可能坏了，这些问题都由 GFS 这个“分布式系统”去考虑解决了。

不过，GFS 仅仅是解决了数据存储和读写的问题。要知道，只是把数据读出来和写回去，只能做做数据备份，这可解决不了什么具体、有意义的问题。所幸，在 GFS 这个分布式文

件系统之上，谷歌又在 2004 年发表了 MapReduce 的论文，也就是一个分布式计算的框架。

那么，我们今天就一起来看看，MapReduce 到底是要解决什么样的问题。而要解决这些问题的系统，又应该要怎么设计。

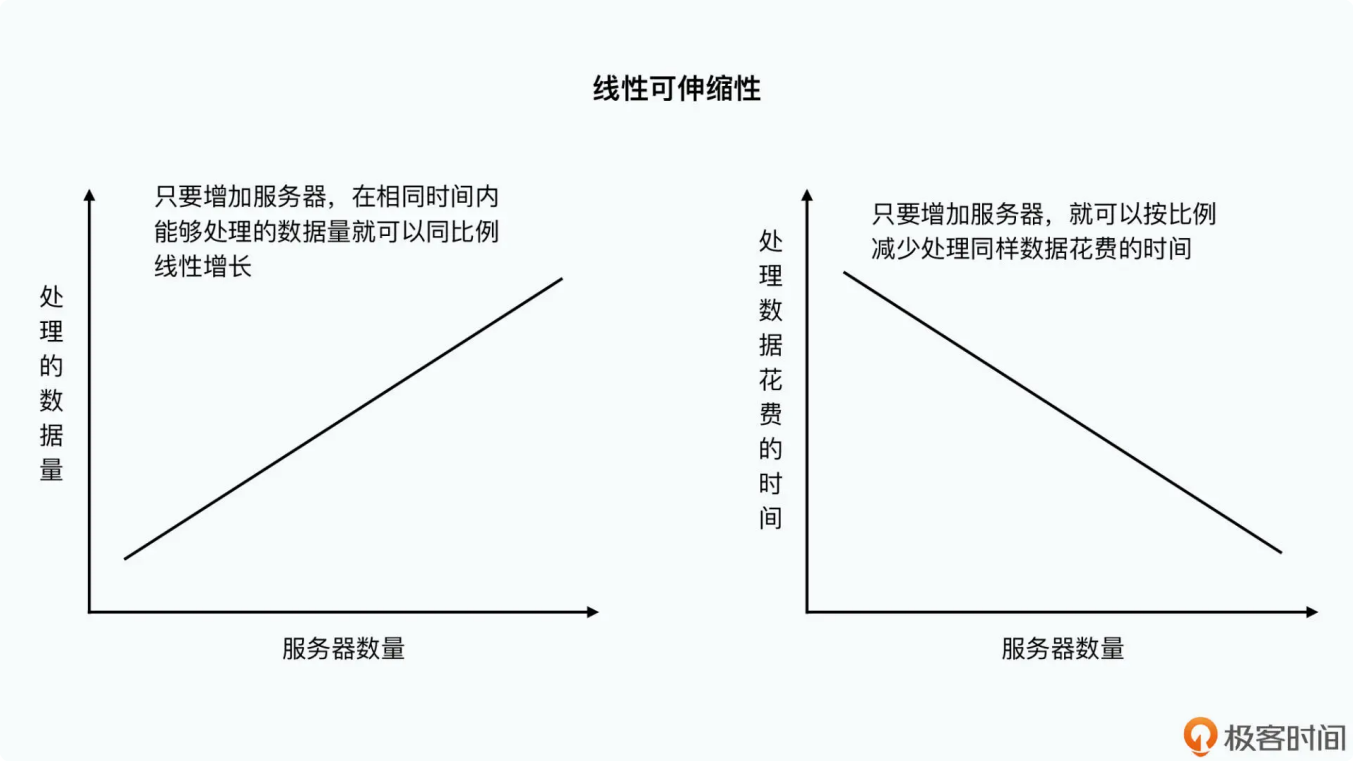
当你仔细了解 MapReduce 的框架之后，你会发现 MapReduce 的设计哲学和 Unix 是一样的，叫做**“Do one thing, and do it well”**，也就是每个模块只做一件事情，但是把这件事情彻底做好。

在学完这两讲之后，你不仅应该了解什么是 MapReduce，MapReduce 是怎么设计的。更重要的是理解如何对系统进行抽象并做出合理的设计。在未来你自己进行系统设计的时候，为各个模块划分好职责和边界，把“Do one thing, and do it well”落到实处。

分布式系统的首要目标：开发人员不懂分布式

如果让你来设计一个分布式数据处理系统，你觉得哪些特性在第一个版本里是特别重要的呢？

我相信答案可能是丰富多彩的。有些人可能会选择“性能”（Performance），分布式系统本来就是因为单机性能不够了，那么把性能视作重要的因素是理所当然的。有些人可能会选择“可伸缩性”（Scalability），只要增加更多的硬件节点，就能处理更大规模的数据，更快地处理原来的数据，那么我们就可以“性能不够，机器来凑”。



线性可伸缩是一个理想情况，实践中是难以达到的

这些答案都没有错，不过，大部分人往往会漏掉一点，那就是“**易用性**”。

更具体一点，就是我们希望来使用这个分布式数据处理系统的人，最好意识不到“分布式”的存在。来使用这个系统的人，只需要撰写处理数据的业务逻辑代码，而不需要关心代码会在哪一台机器上执行、怎么去进行网络通信。撰写的代码只需要解决自身的工作，那就是数据处理的业务逻辑。也就是我们自己的代码，只需要有**“单一职责”**，而不是需要全盘考虑分布式系统的设计问题。

因为分布式系统中会遇到的这些故障和失败，是一个很常见的问题，但是这些问题并不容易处理。我们之所以开发 MapReduce 这样的系统，就是为了让没有分布式系统知识和经验的人，一样可以快速简便地去利用 MapReduce 处理海量的数据。

那么，整个 MapReduce 的论文，其实就是围绕着这一核心点来讲述和设计的。这篇论文，基本上可以看作是三个部分：

1. MapReduce 的计算模型和应用场景；
2. MapReduce 实际是如何实现的，使得开发者无需关心分布式的存在；
3. 如何逐步迭代优化 MapReduce 的性能。

MapReduce：一个分布式的 Bash

实际上，在 2021 年的今天，我们回头看 2004 年发布的 MapReduce，是简单到可以说是简陋的。这也是为什么后续不断有各种各样的新系统出现，比如 Spark、Storm、Dremel 等。但是，尽管简单，MapReduce 的设计其实非常干净利落，它从一开始就是奔着让开发者对“分布式”无感而去的。

那么今天这一讲，我们重点来看一看第一点，也就是 MapReduce 的计算模型和应用场景是什么，然后我们再和 Unix 下的管道做一个对比，看一看为什么说 MapReduce 继承了 Unix 的设计思想。

不过在直接看论文之前，我想请你先自己想一想，**要针对存放在上千个节点的 GFS 上的数据，进行数据处理，你会怎么做？你会有哪些计算方式的需求呢？**

我想了一下，最直接的方式就是在很多台机器上，同时来做运算，也就是进行**并行计算**，这样可以利用我们有上千个节点的优势。而需要的计算方式，抽象来说，无非是三种情况。

第一种，是对所有的数据，我们都只需要单条数据就能完成处理。比如，我们有很多网页的内容，我们要从里面提取出来每一个网页的标题。这样的计算可以完全**并行化**。

第二种，是需要汇总多条数据才能完成计算。比如，要统计日志里面某个 URL 被访问了多少次，只需要简单累加就可以了。或者我们需要更复杂一些的操作，比如统计某个 URL 下面的唯一用户数。而对于这里的第二种情况，我们就需要将所有相同 URL 的数据，□搬运到同一个计算节点上进行处理。不过，在搬运之后，不同的 URL 还是可以放到不同的节点进行处理的。

第三种，自然是一、二两种情况的组合了。比如，我们先从网页数据里面，提取出网页的 URL 和标题，然后根据标题里面的关键字，统计特定关键字出现在多少个不同的 URL 里面，这就需要同时采用一二这两种情况的操作。

当然，我们可以有更复杂的数据操作，但是这些动作也都可以抽象成前面的两个动作的组合。因为无非，我们要处理的数据要么是完全独立的，要么需要多条数据之间的依赖。实际上，前面的第一种动作，就是 MapReduce 里面的 Map；第二种动作，就是

MapReduce 里面的 Reduce；而搬运的过程，就是 Shuffle（混洗，这个概念稍后我会给你介绍）。

那么接下来，我们就一起具体看看 MapReduce 是怎么回事儿。

MapReduce 编程模型

MapReduce 的编程模型非常简单，对于想要利用 MapReduce 框架进行数据处理的开发者来说，只需要实现一个 Map 函数，一个 Reduce 函数。

Map 函数，顾名思义就是一个映射函数，它会接受一个 key-value 对，然后把这个 key-value 对转换成 0 到多个新的 key-value 对并输出出去。

```
1 map (k1, v1) -> list (k2, v2)
```

复制代码

Reduce 函数，则是一个**化简**函数，它接受一个 Key，以及这个 Key 下的一组 Value，然后化简成一组新的值 Value 输出出去。

```
1 reduce (k2, list(v2)) -> list(v3)
```

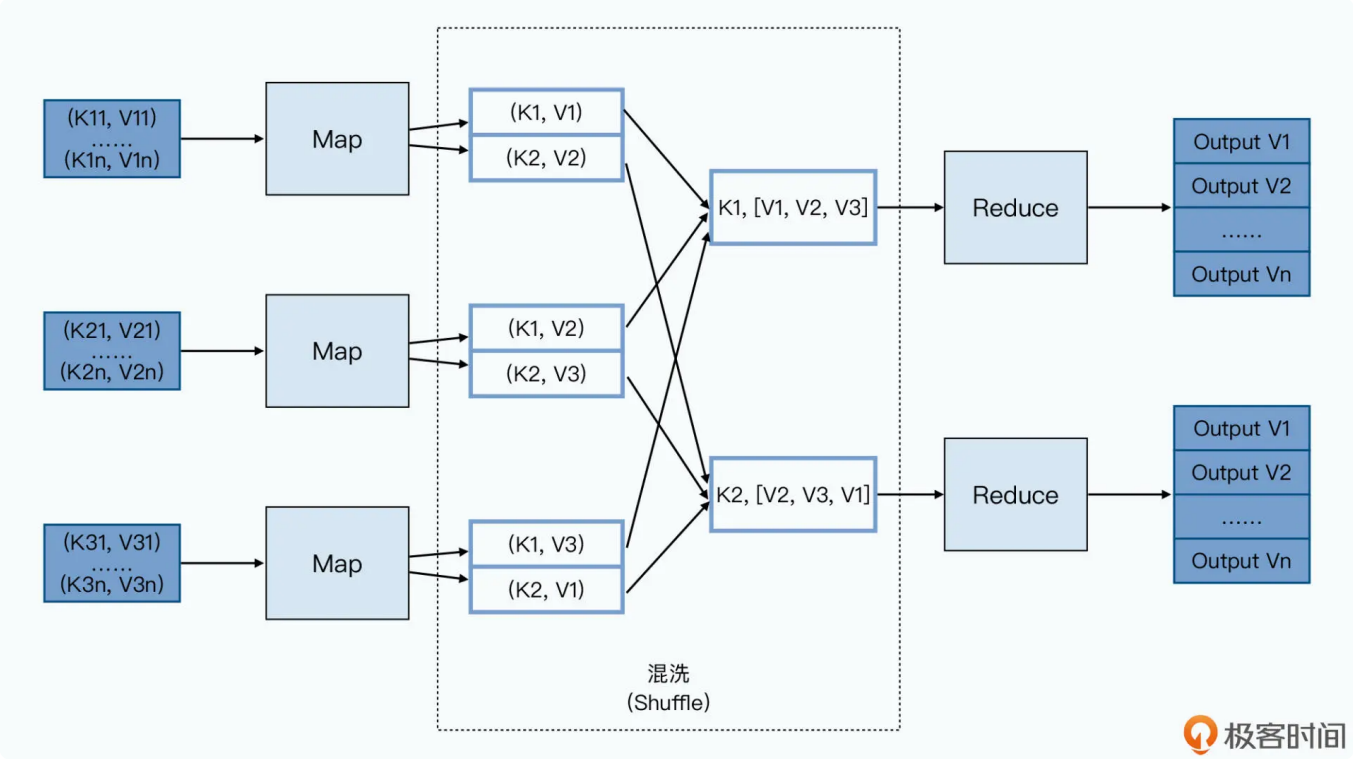
复制代码

而在 Map 函数和 Reduce 函数之外，开发者还需要指定一下输入输出文件的路径。输入路径上的文件内容，会变成一个个键值对给到 Map 函数。而 Map 函数会运行开发者写好的映射逻辑，把数据作为新的一组键值对输出出去。

Map 函数的输出结果，会被整个 MapReduce 程序接手，进行一个叫做**混洗**的操作。混洗会把 Map 函数输出的所有相同的 Key 的 Value 整合到一个列表中，给到 Reduce 函数。并且给到 Reduce 函数的 Key，在每个 Reduce 里，都是按照 Key 排好序的。

这里排好序并不是 MapReduce 框架本身的核心需求，而是为了技术上实现方便。因为我们要把相同 Key 的数据放到一起处理，而通过一个 HashMap 把所有的数据放在内存里又不一定放得下。那么利用硬盘进行外部排序是一个最简单的，没有内存大小依赖的对数据

根据 Key 进行分组的解决办法。最后，在拿到混洗完成，分好组的数据之后，Reduce 函数就会运行你写好的化简逻辑，最终输出结果。



Map输出的相同Key的Value会通过混洗，合并到一起给到Reduce函数

如果你熟悉设计模式的话，你会发现在实现一个 MapReduce 程序上你需要做的事情，其实就是一个典型的模版方法模式（Template Method Pattern）。而 MapReduce 与其说是一个分布式数据处理系统，不如说是**分布式数据处理框架**。因为 MapReduce 框架已经设定好了整个数据处理的流程，你只需要实现 Map 和 Reduce 这两个接口函数，就能完成海量的数据处理程序。

MapReduce 的应用场景

别看在 MapReduce 框架下，你只能定义简简单单的一个 Map 和一个 Reduce 函数，实际上它能够实现的应用场景，论文里可列了不少，包括以下六个：

1. 分布式 grep；
2. 统计 URL 的访问频次；
3. 反转网页 - 链接图；
4. 分域名的词向量；
5. 生成倒排索引；

6. 分布式排序。

下面，我们就主要来关注一下前两个场景的用途，看看最简单的两个场景是如何通过 MapReduce 来实现的。然后，我们可以把其中的第二个场景和 **Unix 下的 Bash 脚本**对应起来，来理解为什么我说 MapReduce 的设计思想，就是来自于 Unix 下的 Bash 和管道。

至于剩下的更复杂的场景，你可以自己试着思考看看，哪些场景可以怎么通过 MapReduce 或者 Bash 来实现。

分布式 grep

在日常使用 Linux 的过程中，相信你没少用过 grep 这个命令。早年间，在出现各种线上故障的时候，我常常会通过 grep 来检索各种应用和 Web 服务器的错误日志，去排查线上问题，如下所示：

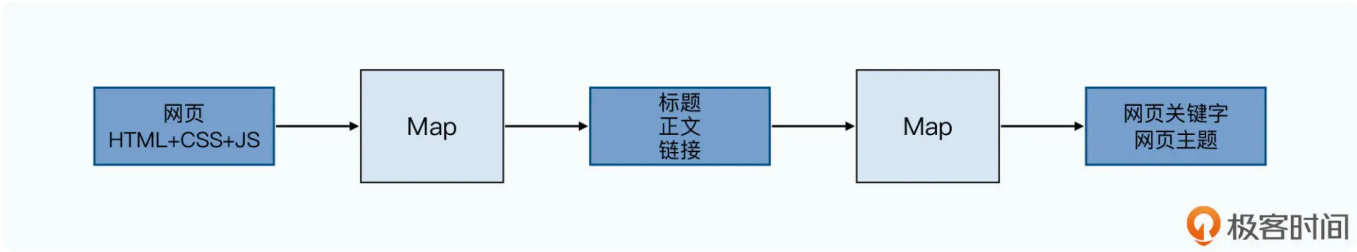
```
1 grep "error" access.log > /tmp/error.log.1
```

复制代码

在单台 Linux 服务器上，我们当然可以用一个 grep 命令。那么如果有很多台服务器，我们怎么才能知道在哪台机器上会有我们需要的错误日志呢？

最简单的办法，当然就是在每台服务器上，都执行一遍相同的 grep 命令就好了。这个动作就是所谓的“分布式 grep”，在整个 MapReduce 框架下，它其实就是一个只有 Map，没有 Reduce 的任务。

在真实的应用场景下，“分布式 grep”当然不只是用来检索日志。对于谷歌这个全球最大搜索引擎来说，这是完美地用来做**网页预处理**的方案。通过网络爬虫抓取到的网页内容，你都可以直接存到 GFS 上，这样你就可以撰写一个 Map 函数，从 HTML 的网页中，提取网页里的标题、正文，以及链接。然后你可以再去撰写一个 Map 函数，对标题和正文进行关键词提取。



这些一步步的处理结果，还会作为后续的反转网页 - 链接图、生成倒排索引等 MapReduce 任务的输入数据。

实际上，**“分布式 grep”就是一个分布式抽取数据的抽象**，无论是像 grep 一样通过一个正则表达式来提取内容，还是用复杂的算法和策略从输入中提取内容，都可以看成是一种“分布式 grep”。而在 MapReduce 这个框架下，你只需要撰写一个 Map 函数，并不需要关心数据存储在具体哪台机器上，也不需要关心哪台机器的硬件或者网络出了问题。

统计 URL 的访问频次

了解了分布式 grep，我们再来看看统计访问频次。这里，我们先以极客时间的专栏用户访问日志作为例子，来看看 MapReduce 可以怎么来统计访问频次。

下面放了一个表格，我们把它叫做 url_visit_logs 。在这个表格里面有三个字段，分别是：

1. **URL**，记录用户具体是看专栏里的哪一篇文章；
2. **USER_ID**，记录具体是哪一个用户访问；
3. **VISIT_TIME**，记录用户访问的具体时间。

URL	USER_ID	VISIT_TIME
https://time.geekbang.org/column/article/91427	USER_0001	2021-02-28 18:12:34+0800
https://time.geekbang.org/column/article/91793	USER_0321	2021-03-05 12:12:34+0800
https://time.geekbang.org/column/article/118016	USER_0001	2021-05-01 09:13:05+0800
.....	.....	.....
https://time.geekbang.org/column/article/89251	USER_3256	2021-07-05 15:28:34+0800

极客时间

数据表url_visit_logs

那么，作为像谷歌这样的搜索引擎，它通常都会有统计网页访问频次的需求。访问频次高的网页，通常可以被认为是内容质量高，会在搜索结果的排名里面，排在更前面的位置。

如果只是极客时间的网页，我们可以把这张表里面的数据放在数据库里面，通过一句 SQL 就可以完成了：

复制代码

```
SELECT URL, COUNT(*) FROM url_visit_logs GROUP BY URL ORDER BY URL
```

但是，如果考虑全网的所有数据网页访问日志，数据库就肯定放不下了。我们可以把这些日志**以文件的形式**放在 GFS 上，然后通过 MapReduce 来做数据统计。

Map 函数很简单，它拿到的输入数据是这样的：

1. Key 就是单条日志记录在文件中的行号；
2. Value 就是对应单条记录的字符串，不同字段之间以 Tab 分割。

Key	当前记录的行号	2132567
Value	字符串，Tab分割字段	https://time.geekbang.org/column/article/91793\tUSER_0321\t2021-03-05 12:12:34+0800

极客时间

Map函数的输入

Map 函数只需要通过一个 split 或者类似的函数，对 Value 进行分割，拿到 URL，然后输出一个 List 的 key-value 对。在当前的场景下，这个 List 只有一个 key-value 对：

1. 输出的 Key 就是 URL；
2. 输出的 Value 为空字符串。

Key	URL	https://time.geekbang.org/column/article/91793
Value	''	''

极客时间

Map函数的输出

这个 URL 肯定不只被访问了一次，因为 MapReduce 框架会把所有相同 URL 的 Map 的输出记录，都混洗给到同一个 Reduce 函数里。所以在这里，Reduce 函数拿到的输入数据是这样的：

1. Key 就是 URL；
2. 一个 List 的 Value，里面的每一项都是空字符串。

Key	URL	https://time.geekbang.org/column/article/91793
list<Value>	['','','','']	['','','','']

极客时间

Reduce 函数的逻辑也非常简单，就是把 list 里面的所有 Value 计个数，然后和前面的 Key 拼装到一起，并输出出去。Reduce 函数输出的 list 里，也只有这一个元素。

list<Value>	[URL + 计数]	['https://time.geekbang.org/column/article/91793\t4']

极客时间

这样一个 MapReduce 的过程，就和我们之前通过 SELECT + GROUP BY 关键字执行的 SQL 起到了同样的作用。

> 其中的 Map 过程，类似于 SELECT 关键字，从输入数据中提取出需要使用的 Key 和 Value 字段。
>
> MapReduce 框架完成的混洗过程，类似于 SQL 中的 GROUP BY 关键字，根据相同的 Key，把 Map 中选择的数据混洗到一起。
>
> 最后的 Reduce 过程，就是先对混洗后数据中的 Value，执行了 COUNT 这个函数，然后再将 Key 和 COUNT 执行的结果拼装到一起，输出为最后的结果。

事实上，SQL 是一种申明式的语言，MapReduce 是我们的实现过程，后面我们在讲解 Hive 论文时候，你就会发现，Hive 的 HQL 就是通过一个个 MapReduce 程序来实现的。而前面的整个 MapReduce 的过程，其实用一段 Bash 代码也可以实现。

复制代码

```
cat $input |
   awk '{print $1}' |
   sort |
   uniq -c > $output
```

在这段 Bash 代码中：

cat 相当于我们 MapReduce 框架从 HDFS 读取数据；

awk 的脚本，是我们实现的 Map 函数；

sort 相当于 MapReduce 的混洗，只是这个混洗是在本机上执行的；

而最后的 uniq -c 则是实现了 Reduce 函数，在排好序的数据下，完成了同一 URL 的去重计数的工作。

如果和 MapReduce 框架对照起来，你会发现：

读写 HDFS 文件的内容，对应着 cat 命令和标准输出；

对于数据进行混洗，对应着 sort 命令；

整个框架，不同阶段之间的数据传输，用的就是标准的输入输出管道。

那么，对于开发者来说，只要自己实现 Map 和 Reduce 函数就好了，其他都不需要关心。而对于实现 MapReduce 的底层框架代码，也可以映射到读取、外部排序、输出，以及通过网络进行跨机器的数据传输就好了。**在这个设计框架下，每一个组件都只需要完成自己的工作，整个框架就能很容易地串联起来了。**

小结

通过这节课，我给你介绍的两个典型的 MapReduce 例子，相信你现在对如何通过 MapReduce 这个分布式框架，来进行数据处理就已经了解清楚了。

作为一个框架，MapReduce 设计的一个重要思想，就是**让使用者意识不到“分布式”这件事情本身的存在**。从设计模式的角度，MapReduce 框架用了一个经典的设计模式，就是模版方法模式。而从设计思想的角度，MapReduce 的整个流程，类似于 Unix 下一个个命令通过管道把数据处理流程串接起来。

MapReduce 的数据处理设计很直观，并不难理解。Map 帮助我们解决了并行在很多台机器上处理互相之间没有依赖关系的数据；而 Reduce 则用来处理互相之间有依赖关系的数据，我们可以通过 MapReduce 框架自带的 Shuffle 功能，通过排序来根据设定好的 Key 进行分组，把相同 Key 的数据放到同一个节点上，供 Reduce 处理。

而作为 MapReduce 框架的使用者，你只需要实现 Map 和 Reduce 两个函数，并且指定输入输出路径，MapReduce 框架就会帮助你完成整个数据处理过程，**不需要你去关心整个分布式集群的存在**。另外，不仅仅是 MapReduce 的用户，只需要考虑“单一职责”，实现自己的 Map 和 Reduce 函数就好了。即使作为 MapReduce 的框架实现，也能够把数据读取、数据输出、网络传输、数据混洗等模块单独拆出来，实现起来也很容易。

Map 和 Reduce 这两个函数虽然非常简单，但是对于输入输出的格式，以及内部具体的逻辑代码没有任何限制，是完全灵活的，足以完成从日志分析、网页处理、数据统计，乃至于搜索引擎的索引生成工作。

事实上，我们在论文中也可以看到，谷歌在多种不同的场景中，都使用了 MapReduce，包括：

大规模的机器学习问题；

谷歌新闻和 Froogle 商品的聚类；

抽取数据生成热门搜索的报表；

大规模的图计算。

这些复杂的问题，都可以通过一个或者多个 MapReduce 的任务的串联来实现。

那么，相信到这里，你已经迫不及待想要知道 MapReduce 底层是怎么实现的了，它是怎么能让开发者不用去关心下面的分布式系统的存在的呢？别着急，在下一讲，我会带你深入到 MapReduce 框架本身的实现中去。

推荐阅读

1. MapReduce 是一个典型的模版方法模式，你可以去读一读《设计模式》这本书的 5.10 小节。事实上，在整个大数据领域中，我们处处可以看到各种不同设计模式的身影。
2. MapReduce 在发明之后，也被用作大规模的机器学习，一个最常用的算法就是通过 L-BFGS 来进行逻辑回归的模型训练。如果你对通过 MapReduce 进行机器学习感兴趣，可以去读一下论文“Large-scale L-BFGS using MapReduce”。

思考题

在前面的 URL 访问频次的应用场景中，我们只是简单地做了单个 URL 的访问频次统计。如果我们想提出两个新需求，一个是不再按照 URL 统计访问频次，而是按照域名统计，第二个是统计结果要按访问的人数多少从高到底排序，我们应该怎么做呢？这个逻辑我们也能通过 Bash 来实现吗？

欢迎在留言区分享你的答案和思考过程，我们一起交流讨论。也欢迎你把今天的内容，分享给更多的朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | The Google File System （三）： 多写几次也没关系

下一篇 07 | MapReduce（二）：不怕失败的计算框架

精选留言 (6)

写留言

Monroe He

2021-10-05

问题2是否可以通过两个mapreduce实现：
mapreduce1: map函数 key(URL+userID): value('1') -> reduce函数去重之后的 key(URL+userID): value('1')
mapreduce2: map直接使用mapreduce1的输出， reduce 函数计数 key(URL):value(sum())

展开

那时刻

2021-10-03

第二个问题，URL 访问统计结果要按访问的人数多少从高到底排序。为了减少Reduce 函

数去重的时候，用户数据量比较大的问题。可以通过两次map reduce操作来减少这个数据量。

1. map以(URL+UserId, 1) 作为输出,表示某个url被某个user访问过一次，reduce输出则...

展开

webmin

2021-10-03

看到MapReduce我就会不自主的映射为递归，Map是递的过程，一路向下进行切分，Reduce是归触底向上一路进行收集。
思考题：
第二个问题与网上流传的在单机上2GB内存排序10GB大小文件的题目类似，需要通过分桶缩小每次处理的大小，按Count数划分区间，每个区间一个文件，然后再对每一个区间进...

展开

qinsi

2021-10-02

想到个有些复杂的方法：

第一轮：map输出<URL+USER_ID, ">，reduce时对value计数，可以得出每个url被每个用户访问了多少次<URL+USER_ID, visitsByUser>

...

展开

leslie

2021-10-02

老师确实把mapreduce拆解的非常合理，这种合理不是简单的应用层的去拆；纵观数据系统这么多年真正合理的都是简单方便；SQL的特点就是简练-这是为何至今四十余年依然未曾衰老。

在路上

2021-10-02

徐老师好，晚上试着读了下MapReduce的论文，比GFS的论文要简单很多。
问题一，如何按照域名统计访问频率？只需要把原来Map函数的输出从List<URL, “”>，变成List<domainName(URL), “”>。
问题二，如何按照访问人数对统计结构排序？假如还是统计URL的访问情况，Map函数的输

出为List<URL, USER_ID>，Reduce函数对USER_ID去重，获得每个URL的访问人数。Re…

展开

作者回复: 第一个问题没有问题。但是第二个问题有一个挑战，就是在“Reduce”函数去重这里，具体怎么去重呢？ 因为如果用户数量足够多，我们会遇到一种情况就是Reduce用内存来进行去重内存可能不够用，再想想。

1